



944-005,015
Serial Number 10/688,210

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Re Application of :
Michael Przybilski, et al :
Serial No. 10/688,210 : Examiner: Junchun Wu
Filed: October 17, 2003 : Group Art Unit: 2196
For: SOFTWARE UPDATING PROCESS FOR MOBILE DEVICES

Director
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

AFFIDAVIT/DECLARATION UNDER 37 CFR 1.131

I, Mika Leppinen, being duly sworn, depose and say:

1. I reside at Haltijatontuntie 31D, 02200 Espoo, Finland.
2. I am one of the inventors named in the U.S. Patent Application Serial No. 10/688,210 filed October 17, 2003.
3. The invention was reduced to practice as a working prototype before November 12, 2002.
4. We are in the possession of the report/working document which incorporates the system described in the present invention (U.S. Patent Application Serial No. 10/688,210) and implemented and tested by Bitfone Company using said invention of ours. The document is dated October 2002. The content of the report is confidential and proprietary to Bitfone Corporation and may not be reproduced, published,

944-005.015
Serial No. 10/688,210

or disclosed to others without the prior written consent of
Bitfone.


Mika Leppinen

Document ID:		
Status:*	Final	
Version:	1.0	
Author:	Bitfone/Iyad Qumei	
Reviewed to a Proposal:	Fred Leland	
Approved:	Harri Okkonen	



mProve Download Agent Integration Code Architecture For The Nokia 7650

*We use the following classification of deliverables:

Draft:	Unfinished document representing author's views
Proposal:	Reviewed by the project manager, represents the views of the project group.
Final:	Deliverable that has formally approved by the customer of the project

Revision History

Approved	Date	Document Version	Description	Author
	10/4/02	0.1	Initial Draft	Iyad Qumei
	10/11/02	0.2	Added new images to user interface section. Documented the sockets module. Formatted table for readability	Iyad Qumei
	10/14/02	0.3	Added new material to the implementation section. Modified format and organization of the section	Iyad Qumei
	10/15/02	0.4	Modified documentation for user guide. Expanded the architecture section to include wrappers implementation. Made editorial changes Added new material in the implementation section. Added overview sections, modified existing material presentation and re-organized the sections.	Iyad Qumei
	10/15/02	0.5	Made editorial changes to the implementation sections.	Iyad Qumei
	10/17/02	1.0	Final formatting changes.	Jennifer Jones

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

1	PURPOSE	5
2	SCOPE	5
3	USER GUIDE.....	6
3.1	Launching the Download Agent Application.....	6
3.2	Main View	7
3.3	Server Information Dialog	7
3.4	Download Update View.....	8
3.5	About View.....	9
3.6	Error Reporting	10
4	MPROVE DOWNLOAD AGENT ARCHITECTURE	12
4.1	Major Components	12
4.2	Download Agent Engine Module.....	14
4.3	Communications Module	15
4.4	User Interface Module.....	15
4.5	Core Download Agent.....	16
4.6	Wrapper Functions Implementation	17
4.6.1	User Interface Wrapper Functions.....	17
4.6.2	Memory Manager Wrapper Functions.....	17
4.6.3	Protocol Settings Wrapper Functions.....	18
4.6.4	Flash Definition Wrapper Functions	18
4.6.5	Device Definitions Wrapper Functions	18
4.6.6	Bearer Definitions Wrapper Functions.....	20

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5	IMPLEMENTATION.....	21
5.1	Graphical User Interface	21
5.1.1	Overview of Graphical User Interface module.....	21
5.1.2	CmProveAppUi.....	25
5.1.3	CmProveViewMain.....	27
5.1.4	CmProveViewDagent	28
5.1.5	CmProveViewAbout	31
5.1.6	CmProveDialogMain	32
5.1.7	CmProveDialogIPEditor	33
5.1.8	CmProveContainerDagent	35
5.1.9	CmProveDialogAbout	37
5.2	mProve Download Agent Engine	38
5.2.1	Overview of mProve Download Agent Engine.....	38
5.2.2	CBFDagentEng	40
5.2.3	CBFAgentActive	43
5.2.4	Update Request States	45
5.2.5	CBFLogfile.....	47
5.2.6	BF_DaGlobals	48
5.2.7	Simulated Flash and RAM.....	51
5.2.8	MUINotify	52
5.3	MProve Download Agent Core	53
5.4	Communications	54
5.4.1	Overview of Sockets Communications	54
5.4.2	CSockets Engine	56
5.4.3	CSocketsWrite.....	60
5.4.4	CSocketsRead	62
5.4.5	CTimeOutTimer.....	65
5.4.6	MTimeOutNotify	66

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

1 PURPOSE

This document describes the integration effort of the mProve Download Agent into the Symbian operating system running on the Nokia 7650 phone.

2 SCOPE

This document outlines and discusses the software architecture of the integration process. It presents details on the implementations for the download agent engine, the communications module, the graphical user interface, the wrapper functions definitions and various changes to the download agent core.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

3 USER GUIDE

This section presents the download agent application user guide. It describes the user interface with the aid of various screen shots and control settings. The order of the presentation follows the logical flow of the download process.

In the user interface design follows the suggested guidelines for Nokia 7650 application development. Therefore, the download agent application has the look and feel of a typical application

3.1 Launching the Download Agent Application

The download agent resides in the main application area. The mProve icon identifies the application for selection and launching. The figure below shows the mProve icon as it appears on the phone's screen and the corresponding control buttons.

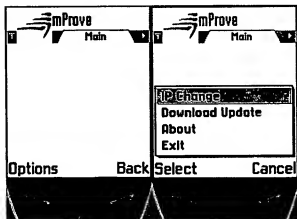


The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

3.2 Main View

The main view is the first to appear on the screen after the application launch. The control buttons provide the options to move to other views. These views include the IP Change view, the Download Update view, and the About view. The Exit and Back controls terminate the application. The figures below show the main view, and the options available for selection.



3.3 Server Information Dialog

The IP Change dialog presents the user with modifiable fields to edit the server name and port number. The OK button accepts the changes and moves to the Download Update view. The Cancel option terminates the application. The figure below shows the IP Change view and associated control buttons.



3.4 Download Update View

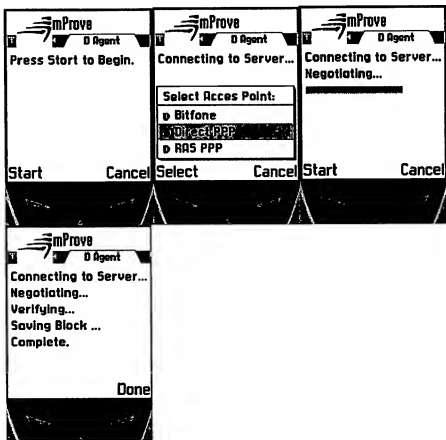
The download agent view presents the user with two control options only. Start initiates the update process, and automatically brings up the access point selection dialog. Cancel terminates the application. The user may cancel the application at any time.

The different stages of the download process appear on the screen as text messages such as Negotiating, Verifying, Saving Block, and Complete. The progress bar keeps track of the time duration for the update package retrieval. Once the retrieval completes, the progress bar disappears and the text messages continue to appear on screen.

Upon completion the user terminates the application with the Done button. The figures below show screen shots of various stages of the download process.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

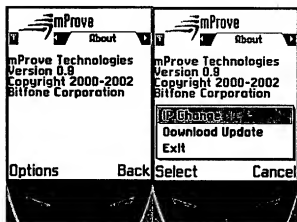


3.5 About View

The About view presents general information about the application. This information includes product name, version number and copyright. Options available for the view guide the user to either IP Change or Download Update view. The figures below show the layout of the view and its options.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.



3.6 Error Reporting

There are two error-reporting mechanisms in the current implementation. First, error messages appear on screen in a separate dialog. The user must select OK to acknowledge the error and continue.

The log file C:\bfdagent.log stores a complete history of the download agent process. This information can be used for debug purposes.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.



The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

4 MPROVE DOWNLOAD AGENT ARCHITECTURE

The objective is to create a download agent application for the Symbian operating system (OS). The application runs on the Nokia 7650 phone. The download agent application integrates the original mProve download agent, referred to as the core, with services provided by the operating system.

The integration process involves several tasks. These tasks are as follows:

- Port the download agent core code to comply with operational requirements by the Symbian OS
- Implement the wrapper functions for the core download agent
- Create the sockets communications module. The communication module facilitates the download of update package from a remote server. A graphical user interface provides high-level control and monitoring of the download process.

4.1 Major Components

The porting process of the core download agent involves modifying the original architecture to comply with the operational requirements of the Symbian OS. The first requirement is to eliminate all modifiable globules referenced by the core agent. The second requirement, which stems from the fact that the Symbian OS is event driven, requires breaking a long running process, such as the download process, into smaller events executed in the proper sequence by the OS.

The download agent Engine module solves these issues. It provides active object to step through the download agent process. States representing the different stages of the download guide the stepping processing. Another engine object includes a data structure that encapsulates all global variables. It initializes the data structure and maintains it throughout its life. Routines that reference global data are modified to accept the object owning the structure as argument.

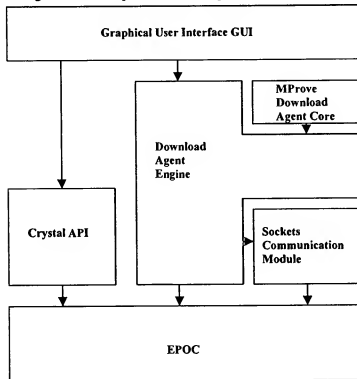
The second component is the communications module. It relies on the Symbian's sockets implementation. This module facilitates the communications between the download agent and a remote update package server.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

The graphical user interface module provides the means for high-level control and monitoring of the download process. It allows the user to initiate the download process, modify the server information, and monitor the download through descriptive messages and other visual effects.

The figure below depicts these components and their relationship with each other and the



operating system. The Download Agent Engine resides on top of the operating system EPOC. (The Symbian OS consists of EPOC and Crystal the graphical API.) The graphical user interface for the download agent uses the Crystal API, and provides services to the download engine. The Sockets engine uses the operating system to provide services to the download engine. The mProve Download Agent Core accesses services through the engine.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

4.2 Download Agent Engine Module

The download agent engine solves the operational requirements related to handling global variables, and making the download process event driven. In addition, it provides a logging mechanism for debug information, and update package storage. The engine provides wrapper functions to pass messages generated by the download agent and the sockets engine to the user interface.

The engine consists of three objects, CBFDAgentEng, CBFDAgentActive and CBFLogfile. The abstraction class MUINotify provides methods to access the user interface for message display and progress update.

All global variables associated with the core download agent are encapsulated in BF_DaGlobals data structures. The structure is initialized as part of creating the CBFDAgentEng object. In turn, routines referencing global variables accept the CBFDAgentEng object as an argument. The consequence of this implementation is to modify all routines utilizing global variables to accommodate an extra argument passed as a pointer to void. The argument is later cast to the appropriate data type for use.

The CBFDAgentEng and CBFDAgentActive objects cooperate to handle the locking problem associated with a long running download process. The CBFDAgentActive class derives from an/the active object base. Active objects respond to events issued by the OS. These events originate from requests initiated by CBFDAgentEng. Hence these two classes form a loop with the help of the operating system. The loop starts by a user request, and ends when the download process completes or the user requests to cancel. The download process is broken into stages. The enumeration TDagentRequest represents all stages.

The Observer class MUINotify is part of the engine module. It defines virtual functions that are the bases to pass messages to the graphical user interface.

A Simulated Flash provides an interim method to storing the update package, and other states that control the download and update processes. The data structure BF_DaSimFlash defines the Flash memory. The BF_DaGlobals encapsulates BF_DaSimFlash. The final solution will

include a native Symbian device driver that resides within the operating system, and is accessible by the agent.

4.3 Communications Module

The architecture for the communications module follows the common implementation for sockets communications. This involves three classes CSocketsEngine, CSocketsRead and CSocketsWrite. These classes control the process of receiving and sending data.

The CSocketsEngine is the main class responsible for creating the communications module. The construction of the CSocketsEngine object includes creating internal timer, collecting server information from user input, opening channel to the socket server, and creating the objects for both CSocketsRead and CSocketsWrite. The CSocketsEngine object provides a method to send data through CSocketsWrite. In addition, it provides access to the received data through the CSocketsRead.

The communications module operates in synchronous mode. The size of data sent and received is set for optimal user interface performance. The engine is responsible for recursively requests sending and receiving data until completion.

In the CsocketRead object, the read buffer is exposed to the download agent core through a shadow buffer and associated methods for reading it. Once the data is received, it gets copied into the shadow buffer. Reading from the shadow buffer is done through methods designed specifically for this purpose. The shadow buffer is circular. It operates with begin and end pointers to facilitate the adding and removing of data.

4.4 User Interface Module

The user interface module provides high-level means for the user to control and monitor the update package download process.

The server name and port number are the only user data inputs required. The remaining controls represent the initiation and the cancellation of the download. In addition, the user has to terminate the application once the download is done.

The monitoring aspect of the user interface involves displaying messages to the user indicating the stage the download agent is in. Further, a progress bar indicates the elapsed time for the download process to complete.

The graphical user interface module follows the standard architecture of a typical Crystal application. This includes applications class CmProveApp derived from CAknApplication, a document class CmProveDocument derived from CAknDocument.

The CmProveAppUi object initializes all user interface views for the download agent, and selects the initial view. The main views are CmProveViewMain, CmProveViewAbout, CmProveViewDagent and CmProveDialogIPEditor.

The main view CmProveViewMain and its dialog CmProveDialogMain provide the entry point for the application. The user can proceed from this view to others as desired.

The CmProveViewAbout and its dialog CmProveDialogAbout provide general information about the application. This information includes application name, version and copyright. The user may select to move to IP Change dialog or Download Update view.

The dialog CmProveDialogIPEditor provides the user with controls to edit the server name and port number. The sockets communication engine accesses this information and stores them internally. The user may select to proceed to Download Update view or cancel the download altogether.

The final view is the heart of the download process. The view CmProveViewDagent and its container CmProveContainerDagent control the start, cancellation and monitor of the download process. They provide utilities to display messages on the screen, or store them in a log file.

4.5 Core Download Agent

Modifications targeting the core download agent resulted from the elimination of global variables, and breaking the long download process into small duration steps.

As mentioned earlier, the data structure BF_DaGlobals encapsulates all modifiable global variables used in the core download agent. The creation of the BFDagentEng results in the creation and initialization of the BF_DaGlobals. The core routines access global variables through passing the BFDagentEng object as an argument.

Breaking the long download process into small duration steps resulted in modification to the da_download_GetDUP () function. The result was several functions. The da_Download_GetDUP_path1 () function is responsible for initializations and checks prior to the download package download process. A typical simultaneous download and save loop is executed within the da_Download_GetDUP_loop1 () function. The CBFDAgentEng controls the recursive calling of the loop body. Finally, the da_Download_GetDUP_Error1 () function is responsible for cleanup due to exceptions occurring during the loop execution.

4.6 Wrapper Functions Implementation

The download agent operation depends on the proper implementation of wrapper functions. The wrapper functions provide parameter definitions for optimal performance. In addition, they provide access to device services. This section describes the implementation of these wrapper functions.

4.6.1 User Interface Wrapper Functions

The DaUI file defines wrapper function to access the screen to display messages from the download agent. In addition, it includes an error message translation table to display readable error messages

4.6.2 Memory Manager Wrapper Functions

The memory manager DaMemory wrapper functions rely on the standard library memory allocation and freeing.

4.6.3 Protocol Settings Wrapper Functions

The DaSettings.C file contains parameters to control the communications protocol. The default parameter seems to work fine. However, changes to the MTU parameter may improve speed, but the 768 limit represents an optimal setting.

4.6.4 Flash Definition Wrapper Functions

The DaFlash DaFlashRAM files define wrapper functions for interacting with the non-volatile memory. The current implementation relies on simulated flash. However, the final implementation will include a device driver for accessing the actual flash memory.

4.6.5 Device Definitions Wrapper Functions

The DaDevice file contains information regarding the device, which is the Nokia 7650 phone. The following table summarizes the type of information accessed by these routines.

DaDevice wrapper functions description	
da_Device_GetDeviceInfo	To return XML string describing the device. void da_Device_GetDeviceInfo(void *aCBFDAgentEng, char *cDeviceInformation);
da_Device_GetServerAddress	To return server connection address. A server address is defined as a free-form 32 byte string. char *da_Device_GetServerAddress(void *aCBFDAgentEng);
da_Device_GetUpdatePackageAddress	To return the address in Flash memory to use when saving an update package. See the Notes for more specific information. unsigned int da_Device_GetStateAddress(void *aCBFDAgentEng);
da_Device_GetBackupAddress	To return the address in Flash to use for backing up updated Flash banks. unsigned int da_Device_GetUpdatePackageAddress(void *aCBFDAgentEng, unsigned int uiDUPsize, unsigned int

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

DaDevice wrapper functions description

	uiStateSize);
	Obtain the address of the update state.
da_Device_GetStateAddress	unsigned int da_Device_GetBackupAddress(void *aCBFDAgentEng, unsigned int uiSize);
	To obtain the value of the active update state.
da_Device_GetStateActiveValue	unsigned int da_Device_GetStateActiveValue(void);
	unsigned int da_Device_Yield(void);
	unsigned int da_Device_GoOffline(void);
	void da_Device_KickWatchdog(void);
	void da_Device_Reset(void);
	void da_Device_Sleep(void *aCBFDAgentEng, int milliseconds);
	bfuquad da_Device_GetMillisecondTick(void *aCBFDAgentEng);
da_Device_SimultaneousDownloadAndSave	To obtain the option of the simultaneous download and save bool da_Device_SimultaneousDownloadAndSave(void);
	To obtain the size of the allocated buffer for the simultaneous download and save. This function call is valid only if da_Device_SimultaneousDownloadAndSave() returns true
da_Device_AllocatedDownloadBufferSize	unsigned int da_Device_AllocatedDownloadBufferSize(void);
	bool da_Device_PackagePlacementForward(void);
	unsigned int da_Device_BatteryStatus(void);
	To save the current download status into the non-volatile memory. The download status will be used if a lost connection needs to be resumed.
da_Device_SaveDownloadStatus	void da_Device_SaveDownloadStatus(void *aCBFDAgentEng,

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

DaDevice wrapper functions description

	char *ptrMem, unsigned int uiSize);
	To obtain the download status from the non-volatile memory.
da_Device_SaveDownloadStatus	int da_Device_RetrieveDownloadStatus(void *aCBFDAgentEng, char *ptrMem, unsigned int uiSize);

4.6.6 Bearer Definitions Wrapper Functions

The DaBearer file contains a routine for selecting the bearer type, and issuing the proper instructions to interact with it. The bearer type for this implementation is TCP/IP. The bearer wrapper functions provide access to the sockets engine module through intermediate mediate routines. The bfbearerlib_tcpip.c and bfcomdrv_tcpip.cpp files define the intermediate layers. These intermediate steps are for the purpose of maintaining consistency with the original download agent implementation.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5 IMPLEMENTATION

5.1 Graphical User Interface

5.1.1 Overview of Graphical User Interface Module

The graphical user interface consists of several objects representing the different views shown to the user. These views provide the means to control and monitor the download process.

The agent application follows the standard Crystal structure for user interface. It includes an object for the application class CmProveApp derived from CAknApplication, and an object for a document class CmProveDocument derived from CAknDocument.

The CmProveAppUi class is responsible for initializing and displaying the different views on the Nokia 7650 phone display. The figure below describes this object and its relationship to the rest of the application. The CmProveAppUi class is derived from CAknViewAppUi. It uses four objects to construct the different views. These objects are CmProveViewMain, CmProveViewDagent, CmProveDialogIPEditor, and CmProveViewAbout.

The CmProveViewMain and CmProveViewAbout classes have dialog objects. These objects are CmProveDialogMain and CmProveDialogAbout, respectively. The view objects are responsible for updating the display with their data. The dialog objects are responsible for storing and maintaining the data associated with each view.

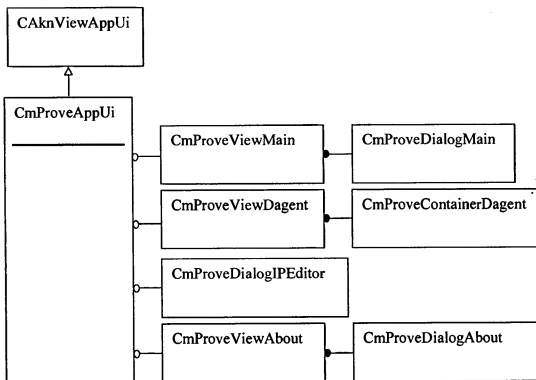
The CmProveDialogIPEditor object allows the user to enter the desired server information. Its implementation is rather unique, where it does not have a view. The server information collected by this dialog is stored directly into the CmProveAppUi object. On the other hand, CmProveAppUi object provides the same information for the CmProveDialogIPEditor to initialize its data upon display.

The CmProveViewDagent class controls the download processes. It is derived from CAknView and MUInotify classes. The MUInotify is an abstract class. It provides through CmProveViewDagent class methods for the download application to display and monitor the

progress of the download process. The view object owns iContainer and iLogfile objects. The iLogfile object provides a mean to store messages for debug purposes.

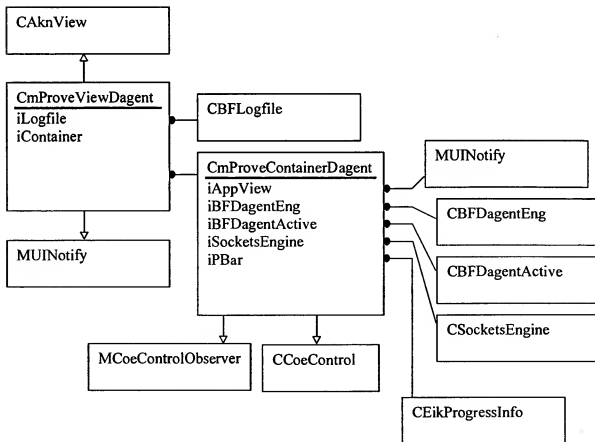
The container class CmProveContainerDagent owns several objects. These objects represent the download agent engine CBFDAgentEng and CBFDAgentActive, and the communications module CSocketsEngine. In addition, the container owns objects for message display and progress bar.

The following diagrams describe the different objects that make up the user interface. The relationship between these objects and other download agent's modules is illustrated.



The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.



The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.2 CmProveAppUi

Class CmProveAppUi : public CAknViewAppUi

An instance of class CmProveAppUi is an object that uses several view objects to create the user interface for the download agent applications.

ConstructL	Performs second phase construction of the object. void ConstructL();
Destructor	Free up resources associated with the object. ~CmProveAppUi()
SetServerNm	Sets the private data member iServer to the desired name identified by aServerNm. void SetServerName(TDesC& aServerNm);
SetPortNumber	Sets the private data member iPort to the desired port number identified by aPortNo. void SetPortNumber(TInt aPortNo)
ServerName	Returns server name associated with private data member iServerNM. TDesC& ServerName();
PortNumber	Return port number associated with private data member iPortNo. TInt PortNumber()
HandleCommandL	Handles user driven events in relationship to user interface. Commands are defined in the resource file. void HandleCommandL(TInt aCommand)
HandleKeyEventL	Handles key events associated with the user interface. virtual TKeyResponse HandleKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType);
iNaviPane	Holds the address of the navigation pan control. CAknNavigationControlContainer* iNaviPane
iDecoratedTabGroup	Holds the address of the navigation decorator. CAknNavigationDecorator* iDecoratedTabGroup

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CmProveAppUi : public CAknViewAppUi

iTabGroup	Holds the address of the tab group, which is read from the resource file. CAknTabGroup* iTabGroup
-----------	--

iServerName	Stores the destination server name. TBuf<KMaxLengthServerName> iServerName
-------------	---

iPortNumber	The destination port number. TInt iPortNumber;
-------------	---

KMaxLengthServerName	Static variable that holds the maximum string length for the server name
----------------------	--

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.3 CmProveViewMain

Class CmProveViewMain : public CAknView

An instance of class CmProveViewMain is an object responsible for the display of the Main view. The data to be displayed are maintained in the dialog object, CmProveDialogMain.

ConstructL	Perform second phase construction of the CmProveViewMain object. void ConstructL()
Destructor	Destroy this object releasing all resources owned by the object. ~CmProveViewMain ()
Id	Returns the view identification number. TUid Id() const;
HandleCommandL	Handle user driven events in relationship to user interface. Commands are defined in the resource file. void HandleCommandL(TInt aCommand)
HandleClientRectChange	Update the view content to match the display screen. void HandleClientRectChange()
DoActivateL	Create and display the Main dialog view object. void DoActivateL(const TVwsViewId& aPrevViewId, TUid aCustomMessageId, const TDcsC8& aCustomMessage)
DoDeactivate	Destroy and hide the Main dialog view object from the screen. void DoDeactivate()
iContainer	Reference to the dialog object of the Main view. CmProveDialogMain* iContainer

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.4 CmProveViewDagent

Class CmProveViewDagent : public CAknView , public MUINotify

An instance of class CmProveViewDagent is an object responsible for the display of the Download Update view. This class has a container class. It owns a CBFLogfile object for storing debug information.

Constructor	Creates CmProveViewDagent object as default constructor CmProveViewDagent();
NewL	Construct a CmProveViewDagent object using two phase construction and returns a pointer to the object. static CmProveViewDagent* NewL();
NewLC	Construct a CmProveViewDagent object using two phase construction, pushes the object onto the cleanup stack, and returns a pointer to the object. static CmProveViewDagent* NewLC();
ConstructL	Performs second phase construction of the CmProveViewDagent object. void ConstructL();
Destructor	Destroy this object and release all resources owned by it. ~CmProveViewDagent();
UpdateCbaL	Alters the definition of the CBA based on the current context. void UpdateCbaL(TInt aResourceId);
Progress Bar Control Utilities	Create, display and destroy utilities for the progress bar. void CreateProgressBarsL(); void DeleteProgressBarsL(); void IncrementBarsAndDraw(TInt increment); void ResetAllValues(); void SetFinalValue(TInt aFinalValue);
Id	Returns the identification number of the download agent view.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CmProveViewDagent : public CAknView , public MUINotify

	TUId Id() const;
HandleCommandL	Handle user driven events in relationship to user interface. The resource file defines these commands. void HandleCommandL(TInt aCommand)
HandleClientRectChange	Update the view content to match the display screen. void HandleClientRectChange()
DoActivateL	Create and display the Main dialog view object. void DoActivateL(const TVwsViewId& aPrevViewId, TUId aCustomMessageId, const TDesC8& aCustomMessage)
DoDeactivate	Destroy and hide the Main dialog view object from the screen. void DoDeactivate()
Write	Wrapper function to container method ShowTextOnScreen. void Write(const TDesC &aMsg);
ClearScreen	Wrapper function to container method ClearScreen. void ClearScreen();
NextCommand	Execute HandleCommandL method using the passed argument. void NextCommand(TInt aCommand);
mtAgent_TestShowResult	Wrapper function to display mTest results on screen. void mtAgent_TestShowResult(TUInt8 aPort, TInt aRetCode, TDes8 &aMsg);
Write to Log File Utilities	Write messages to log file iLogfile. The method overload is for handling different types of arguments void PrintNotify(const TDesC& aDes, TUInt aFontStyle = 0); void PrintNotify(const TDesC8& aDes, TUInt aFontStyle = 0); void PrintNotify(TInt aNumber); void ErrorNotify(const TDesC& aErrorMessage, TInt aErrCode);

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CmProveViewDagent : public CAknView , public MUINotify

	Write messages to screen. The method overloaded is for handling different types of arguments
Write to screen utilities	void PrintToScreen(const TDesC& aDes, TUint aFontStyle = 0); void PrintToScreen(const TDesC&& aDes, TUint aFontStyle = 0); void PrintToScreen(TInt aNumber);
	Write to screen wrapper for the download agent core user interface.
Write utilities for integration with download agent.	void myPrint_char(const char aStr); void myPrint_str(const char *aStr); void myPrint_str_val(const char *aStr, unsigned int aValue); void myPrint_str_2val(const char *aStr, unsigned int aValue1, unsigned int aValue2);
PrintBinary	Write server communications into a log file iDatfile for debug purposes.. void PrintBinary(const TDesC& aDes);
SetStatus	Writes communications module state change into the log file iLogfile. void SetStatus(const TDesC& aStatus);
CancelRequest	Issues cancel request for the communications module. This will initiate the proper shutdown sequence for resources associated with socket communications. void CancelRequest();
iLogfile	The log file object CBFLogfile* iLogfile;
iDatfile	The server communications data file object CBFLogfile* iDatfile
iContainer	Holds pointer to the container of the download agent object. CmProveContainerDagent* iContainer

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.5 CmProveViewAbout

Class CmProveViewAbout : public CAknView

An instance of class CmProveViewAbout is an object responsible for the display of the About view. The container object CmProveDialogAbout maintains the data for the view.

Constructor	Perform second phase construction of the CmProveViewAbout object. void ConstructL()
Destructor	Destroy this object releasing all resources owned by the object. ~CmProveViewAbout()
Id	Returns the view identification number. TUid Id() const;
HandleCommandL	Handles user driven events in relationship to user interface. The resource file defines these commands. void HandleCommandL(TInt aCommand)
HandleClientRectChange	Update the view content to match the display screen. void HandleClientRectChange()
DoActivateL	Create and display the dialog view object. void DoActivateL(const TVwsViewId& aPrevViewId, TUid aCustomMessageId, const TDesC8& aCustomMessage)
DoDeactivate	Destroy and hide the dialog view object from the screen. void DoDeactivate()
iContainer	Stores reference to the container. CmProveDialogAbout* iContainer

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.6 CmProveDialogMain

Class CmProveDialogMain : public CEikDialog

An instance of class CmProveDialogMain is an object that maintains data for the CmProveViewMain view object.

Destructor	Destroy object releasing all resources it owns. ~ CmProveDialogMain ()
PreLayoutDynInitL	Defines the layout of the dialog window before it is displayed. The resource file defines the layout. void PreLayoutDynInitL()
OkToExitL	Called by Symbian framework when the OK button is pressed. TBool OkToExitL(TInt aButtonId)

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.7 CmProveDialogIPEditor

Class CmProveDialogIPEditor : public CAknDialog

An instance of class CmProveDialogIPEditor is an object is responsible for creating and displaying the dialog for the object. The object provides to modify the server name and port number by the user. Modified data are stored in the application user interface object CmProveAppUi.

NewL	Construct CmProveDialogIPEditor object with aAppUi using two-phase constructor and return a pointer to the object. static CmProveDialogIPEditor* NewL(CmProveAppUi& aAppUi)
NewLC	Construct CmProveDialogIPEditor object with aAppUi using two-phase constructor, push the object onto the cleanup stack, and return a pointer to the object. . static CmProveDialogIPEditor* NewLC(CmProveAppUi& aAppUi)
ExecuteLD	Display and run the dialog. Return zero if dialog was cancelled, otherwise returns the ID of button that closed dialog. TInt ExecuteLD()
Constructor	Perform second phase construction of the object. void ConstructL()
Destructor	Destroy this object and free up any resources owned by it. ~CmProveDialogIPEditor()
AppUi	Return a pointer to the Application user interface AppUi object. CmProveAppUi& AppUi() const;
OkToExitL	Called by Symbian framework when the OK button is pressed. TBool OkToExitL(TInt aKeycode);
SetTextL	Copy text described by aText into an edit window type control defined by aControl. void SetTextL(TInt aControl, const TDesC& aText);
SetNumber	Copy number described by aNumber into an edit window type control defined by aControl.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CmProveDialogIPEditor : public CAknDialog

	void SetNumber(TInt aControl, TInt aNumber)
GetText	Copy text defined in an edit window type control aControl into aText. void GetText(TInt aControl, TDes& aText)
GetNumber	Return number defined in an edit window type control aControl. TInt GetNumber(TInt aControl)
	void PreLayoutDynInitL();
SaveSettings	Save server name and port number into data members of CmProveAppUi. void SaveSettings();
iAppUi	Holds a reference to CmProveAppUi object. CmProveAppUi& iAppUi;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.8 CmProveContainerDagent

Class CmProveContainerDagent : public CCoeControl, MCoeControlObserver

An instance of class CmProveContainerDagent is an object for storing and maintaining the data for the CmProveViewDagent view. The container object includes display mechanism for text and progress bar. In addition, the container owns several objects including the download agent engine, and communications module.

ConstructL	Perform second phase construction of the object. void ConstructL(const TRect& aRect, MUINotify &aAppView);
Destructor	Destroy this object releasing all resources it owns. ~CmProveContainerDagent();
ShowTextOnScreen	Display text information about the download state on the screen. The aText argument defines the state. void ShowTextOnScreen(const TDesC& aText);
Print	Wrapper to the ShowTextOnScreen method. void Print(const TDesC& aText);
PrintNewLineL	Advances the current screen position to a new line. void PrintNewLineL();
ClearScreen	Clears screen from all text. void ClearScreen();
IncrementBarsAndDraw	Increments the progress bar by alncrement value. void IncrementBarsAndDraw(TInt alncrement);
ResetAllValues	Resets the progress bar value to zero. void ResetAllValues();
CreateProgressBarsL	Create progress bar object, and display it on the screen. void CreateProgressBarsL();
SetFinalValue	Sets the final value of the progress bar.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CmProveContainerDagent : public CCoeControl, MCoeControlObserver

	void SetFinalValue(TInt aFinalValue);
DeleteProgressBarsL	Destroy the progress bar object, and remove it from the screen. void DeleteProgressBarsL();
SizeChanged	Respond to size changes of component objects. This function is called in part of the Symbian framework. void SizeChanged()
CountComponentControls	Returns the number of controls in a compound control. This function is called in part of the Symbian framework. TInt CountComponentControls() const;
ComponentControl	Returns control from a compound control referenced by its ID. This function is called in part of the Symbian framework. CCoeControl* ComponentControl(TInt aIndex) const;
Draw	Draw the screen with active controls. The windows server calls this function. void Draw(const TRect& aRect) const;
iTextLines	Container for messages to be displayed on screen. RArray <CEikLabel*> iTextLines
iPBar	Stores pointer to progress bar object. CEikProgressInfo* iPBar
iSocketsEngine	Stores pointer to communications engine object. CSocketsEngine* iSocketsEngine
iBFAgent	Stores pointer to download agent engine object. CBFDagentEng* iBFAgent
iBFDagentActive	Stores pointer to download agent active object. CBFDagentActive* iDagentActive

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.1.9 CmProveDialogAbout

Class CmProveDialogAbout : public CEikDialog

An instance of class CmProveDialogAbout is an object to store and maintain data for the CmProveViewAbout object. This information includes name, version and copyright.

Destructor	Destroy this object releasing all resources owned by the object ~CmProveDialogAbout()
PreLayoutDynInitL	Defines the layout of the dialog window before it is displayed. The resource file defines the layout. void PreLayoutDynInitL()
OkToExitL	Called by Symbian framework when the OK button is pressed. TBool OkToExitL(TInt aButtonId)

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2 mProve Download Agent Engine

5.2.1 Overview of mProve Download Agent Engine

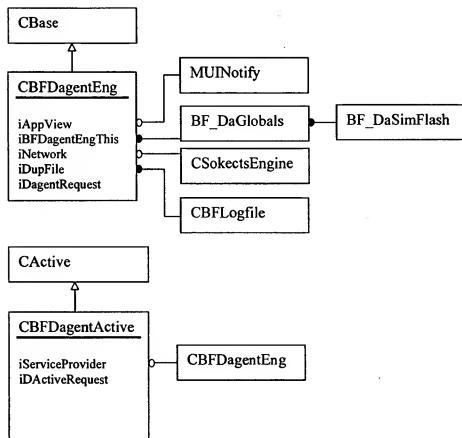
The download agent engine consists of several objects. These are CBFDagentEng, CBFDagentActive, MUINotify and CBFLogfile.

The CBFDagentEng object is derived from Cbase. It owns the global data structure iBFDagentEng_This, the iDupfile object for storing the update package. In addition, the object uses an observer object iAppView and a communications engine object.

The CBFDagentActive object is derived from CActive. It uses a iServiceProvider object of type CBFDagentEng.

Both CBFDagentEng and CBFDagentActive objects control the download process with the aid of internal state trackers iDagentRequest and iDActiveRequest. The enumeration TDagentRequest defines these states.

The following diagram identifies the download agent engine objects. It describes their relationship with each other and the remainder of the application.



The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.2 CBFDDagentEng

Class CBFDDagentEng : public Cbase

An instance of class CBFDDagentEng is an object responsible for executing the download agent process in the proper order. The object owns iBFDagentEng_This global data structure, and iDupfile for storing the update package.

Constructor	Construct a CBFDDagentEng object with CSocketsEngine aNetwork, and observer aAppView. This is the first phase of two-phase object construction. CBFDDagentEng(CSocketsEngine *aNetwork, MUINotify &aAppView);
NewL	Construct a CBFDDagentEng object with CSocketsEngine aNetwork, and observer aAppView using two-phase object construction. Returns a pointer to the object. static CBFDDagentEng* NewL(CSocketsEngine &aNetwork, MUINotify &aAppView)
NewLC	Construct a CBFDDagentEng object with CSocketsEngine aNetwork, and observer aAppView using two-phase object construction. Push the object onto the cleanup stack. Returns a pointer to the object. static CBFDDagentEng* NewLC(CSocketsEngine &aNetwork, MUINotify &aAppView)
ConstructL	Performs second phase construction of the CBFDDagentEng object. void ConstructL();
Destructor	Destroy this object and release all resources owned by it. ~CBFDDagentEng();
RequestTheService	Controls the transition between the different stages of the download process. It works in conjunction with an active object. void RequestTheService(TAgentRequest aDActiveRequest, TRequestStatus& aStatus);
CancelServiceRequest	Cancels the download process, by performing proper cleanup procedure. void CancelServiceRequest();
GetDUP_Done	Returns true if the update package download is complete, otherwise it returns false.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CBFDAgentEng : public Cbase

	TBool GetDUP_Done();
writeDUPfile	Writes the data update package into a file iDupfile. void writeDUPfile(char *iDupBuffer, TUint size);
iBFDAgentEngThis	Holds a pointer to BF_DaGlobals data structure. TAny *iBFDAgentEngThis;
iNetwork	Stores pointer to Communications engine object. CSocketsEngine *iNetwork;
iAppView	Stores reference to Observer. The observer provides mechanism for printing messages, both to screen and log file. MUINotify &iAppView;
iDupFile	Stores pointer to Object of type CBFLogfile for storing the update package. CBFLogfile *iDupfile;
bConnectionEstablished	The state of connection with server. TBool bConnectionEstablished;
uiDownloadSize	The update package size. Tuint uiDownloadSize;
RequestTheServiceErrorFlag	Keeps track of error state during the download process. This is necessary because control shifts between the active object and this object. TBool RequestTheServiceErrorFlag;
iDagentRequest	The state of download agent request. TDagentRequest iDagentRequest;
iTimer	RTimer resource. RTimer iTimer;
timerStatus	Timer request status.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

Class CBFDAgentEng : public Cbase

TRequestStatus timerStatus;

Current time information.

time

TTime time;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.3 CBFAgentActive

class CDAgentActive : public CActive

An instance of class CDAgentActive is an active object responsible for executing the download process. Different stages execute based on events driven by the operating system. The event initiation is performed by the download agent engine object iServiceProvider.

Constructor	Construct CDAgentActive object with observer aServiceProvider. This is the first phase of two-phase object construction. CDAgentActive(CBFDagentEng* aServiceProvider);
NewL	Construct CDAgentActive object with observer aServiceProvider using two-phase object construction. Pushes object onto cleanup stack. Returns pointer to the object. static CDAgentActive* NewL(CBFDagentEng* aServiceProvider);
ConstructL	Performs second phase construction. void ConstructL();
Destructor	Destroy object and release all resources owned by it. ~CDAgentActive();
IssueRequest	Issue request to execute the different stages of the download process. The request is issued for the iServiceProvider object with the appropriate request state. void IssueRequest(TDagentRequest zRequest);
Cancel	Process Cancel request, with appropriate cleanup procedure. Implementation of the virtual Cancel method for the active object. void Cancel();
Canceled	Returns the true if object is already canceled, otherwise returns false. TBool Canceled();
DoCancel	Performs specific procedure pertaining to the cancel request. void DoCancel();
RunL	Controls the execution of different stages of the download process. The framework calls this function, once the active object state changes to pending.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CDAgentActive : public CActive

void RunL();

iServiceProvider

Download agent engine object.

CBFDagentEng* iServiceProvider

iDActiveRequest

Request state for active object.

TDagentRequest iDActiveRequest;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.4 Update Request States

enum zTDagentRequest

Enumeration representing the different states the download agent process goes through. These states coordinate the execution of the download process between the agent engine object and the active object.

The Request State	EBf_AgentStart
	EBf_AgentConnect
	EBf_AgentGetSizeInfo
	EBf_AgentDownloadPackage
	EBf_AgentRelease
	EBf_AgentDisconnect
	EBf_AgentVerify
	EBf_AgentCommit
	EBf_AgentFree
	EBf_AgentSuccess
	EBf_AgentCancel
	EBf_AgentGetDUP_loop1
	EBf_BadDagentRequest
Request Complete State	EBf_BadDagentCancelRequest
	EBf_AgentStart_Complete
	EBf_AgentConnect_Complete
	EBf_AgentGetSizeInfo_Complete
	EBf_AgentDownloadPackage_Complete
	EBf_AgentRelease_Complete
	EBf_AgentDisconnect_Complete
	EBf_AgentVerify_Complete

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

enum zTDagentRequest

	EBf_AgentCommit_Complete
	EBf_AgentFree_Complete
	EBf_AgentSuccess_Complete
	EBf_AgentCancel_Complete
	EBf_AgentGetDUP_loop1_Complete,
	EBf_BadDagentStatus ,
	EBf_AgentStart_Error,
	EBf_AgentConnect_Error,
	EBf_AgentGetSizeInfo_Error,
	EBf_AgentDownloadPackage_Error,
	EBf_AgentRelease_Error,
	EBf_AgentDisconnect_Error,
Request Error State	EBf_AgentVerify_Error,
	EBf_AgentCommit_Error,
	EBf_AgentFree_Error,
	EBf_AgentSuccess_Error,
	EBf_AgentCancel_Error,
	EBf_AgentGetDUP_loop1_Error,
	EBf_BadDagentError

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.5 CBFL logfile

class CBFL logfile : public RFile

An instance of class CBFL logfile is an object, which is used to create instances for log files.

fsSession	File session resource. The session is usually handled by the operating system. However, it was added here for completeness. RFile fsSession;
Fp	File handle resource. RFile fp;
Constructor	Construct a CBFL logfile object. The first stage of two-phase construction. CBFL logfile();
Destructor	Destroy object and releases all resources owned by it. ~CBFL logfile(void);
ConstructL	Performs second phase construction with a file name representing the full path and file name. void ConstructL(TDesC& filename);
Write	Writes to log file TdesC message. void Write(const TDesC& message);
Write	Writes to logfile TdesC8 message. void Write(const TDesC8& message);
ifileName	Buffer holding full qualifying file name. TBuf<256> ifileName;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.6 BF_DaGlobals

struct BF_DaGlobals

An instance of class BF_DaGlobals is an object, which contain all global variables associated with the download agent.

engineSupport	Holds point to CBFDAgentEng object.
	void *engineSupport
Download Agent	bool bStateInitialized
	UpdateStateDescriptor USD
	char *cptrDUP
	unsigned int uiDUPSize
	unsigned int uiDUPCRC
	bool bChannelOpened
	bool bProtocolStarted
	DownloadMEM zDownloadMEM
	unsigned char *cPayload
	unsigned int uiProtocolMTU
Download	char *cptrTest
	bool bDisplayProgressBar;
	unsigned int uiProgressBarSizeElapsed;
Protocol	UIRFields UIRInformation
	unsigned char cCurrentFrameNumber
	unsigned int uiPathMTU
	unsigned char cRecvBuf[RECVBUFFERSIZE]
	unsigned int uiRecvBufIdx
	int iRecvBufLimit

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

struct BF_DaGlobals

	unsigned int uiSleepRIT
CRC	unsigned long *da_crc_table;
	volatile unsigned char *da_heap_heapBegin;
	unsigned long da_heap_ulHeapMemoryBegin;
	bool da_heap_bHeapMemorySPShared;
	unsigned long da_heap_ulHeapMemoryLimit;
Heap	unsigned int da_heap_heapCounter_malloc
	unsigned int da_heap_heapCounter_free
	unsigned int da_heap_heapCounter_realloc
	unsigned int da_heap_heapCounter_calloc
	unsigned int da_heap_heapCounter_totalmem
	unsigned int da_heap_heapCounter_maxmem
	int send_cnt
	int receive_cnt
mTest	int recvbufpos
	int recvbuflimit[5]
	unsigned char SendBufferStr[5][200]
	unsigned char RecvBufferStr[5][200]
	bool da_debug_feedback
Debug	char da_debug_buffer[128]
	bool da_debugram_feedback
	char da_debugram_buffer[128]
Bearer	da_bearer_type da_bearer_selected
Progress Bar	unsigned int progress_full_scale

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

struct BF_DaGlobals

	BfFlashLib_This *bflashlib
Simulated flash, ram	char *FlashSimulated
	char *RAMSimulated
	char *GetDUP_cptrStoreBuffer
da_Download_GetDUP_path1	bool GetDUP_bFirstSegment
	bool GetDUP_Done

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.7 Simulated Flash and RAM

struct BF_DaSimFlash

An instance of struct BF_DaSimFlash is an object representing the simulated Flash and RAM memory. This would be replaced with an interface to writing directly to the flash.

```
unsigned int dt_device_flashbase
unsigned int dt_device_flashsize
unsigned int dt_device_flashblocksize
unsigned int dt_device_flashwaitclock
unsigned int dt_device_stateaddress
unsigned int dt_device_updatepackageaddress
unsigned int dt_device_backupaddress
unsigned int dt_device_updatestatus
unsigned int dt_device_rambase
unsigned int dt_device_ramsize
```

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.2.8 MUINotify

class MUINotify

MUINotify is an abstract class specifying methods for screen display, creating log files and manipulating progress bar.

Writes to log file	virtual void PrintNotify(const TDesC& aMessage, TUint aAttributes = 0) = 0;
	virtual void PrintNotify(const TDesC8& aMessage, TUint aAttributes = 0) = 0;
	virtual void PrintBinary(const TDesC8& aDes) = 0;
	virtual void PrintNotify(TInt aNumber) = 0;
	virtual void ErrorNotify(const TDesC& aErrorMessage, TInt aErrCode) = 0;
Progress Bar Controls	virtual void CreateProgressBarsL() = 0;
	virtual void IncrementBarsAndDraw(TInt increment) = 0;
	virtual void SetFinalValue(TInt aFinalValue) = 0;
	virtual void ResetAllValues() = 0;
SetStatus	Prints the status of communications module.
	virtual void SetStatus(const TDesC& aStatus) = 0;
ClearScreen	Clear text from screen.
	virtual void ClearScreen() = 0;
UpdateCbaL	Updates command buttons
	virtual void UpdateCbaL(TInt aResourceId) = 0;
CancelRequest	Cancel request for download process
	virtual void CancelRequest() = 0;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.3 MProve Download Agent Core

The primary change to the download agent involved changing argument passing to certain functions. These functions access global variables, which are made available through a data structure owned by the download agent engine.

In addition, the `da_Download_GetDUP` function is modified to accommodate event drive framework. The function was modified to perform the package retrieval in multiple steps with shorter time duration. The following table lists the new functions and describes their role in the download process.

Download Agent Process

Changes made to the download agent core to improve responsiveness of the application under event driven architecture of the Symbian operating system

<code>da_Download_GetDUP</code>	Gateway function to separate the process of downloading the update packet into simultaneous download and save, and one shot download. <code>int da_Download_GetDUP(void *aCBFDAgentEng)</code>
<code>da_Download_GetDUP_path1</code>	Initiates the process of download and save of the update packet. <code>int da_Download_GetDUP_path1 (void *aCBFDAgentEng)</code>
<code>da_Download_GetDUP_loop1</code>	Executes a single loop of the download process. This approach places the role of recursive execution to the calling function. <code>int da_Download_GetDUP_loop1(void *aCBFDAgentEng)</code>
<code>da_Download_GetDUP_Error1</code>	Handles exceptions by freeing resources allocated for the download process. <code>int da_Download_GetDUP_Error1 (void *aCBFDAgentEng)</code>

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4 Communications

5.4.1 Overview of Sockets Communications

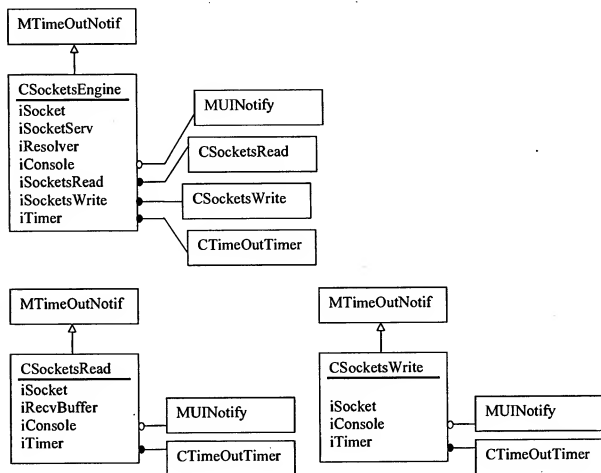
The communications module consists of three main objects. These objects are CSocketsEngine, CSocketsRead and CSocketsWrite.

CSocketsEngine is the main object in the communications module. The object owns the sockets read and write objects. In addition, it owns an iTimer object to keep track of time during communication sessions. The object uses an observer class iConsole to send messages for display and storage in log files.

The CSocketsRead class is responsible for receiving data from the server and making it available to the download agent core. An object of this class owns an iTimer object, and uses iConsole observer for message printing. The object moves the received data into a secondary circular buffer, also known as the shadow buffer. The download agent accesses this buffer to get the required data.

The CSocketsWrite class is responsible for sending data to the server. An object of this class owns an iTimer object, and uses iConsole object for message printing.

The CTimeOutTimer class is derived from CTimer. Its purpose is to keep of track of time for time-outs detection.



The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4.2 CSockets Engine

class CSocketsEngine : public MTimeOutNotify

An instance of class CSocketsEngine is an object responsible for creating the communications module. This includes creating a separate sockets reader and writer objects. In addition, it initializes the receive buffers.

Constructor	Create CSocketsEngine object with observer aConsole using two-phase construction. CSocketsEngine(MUINotify& aConsole)
NewL	Create CSocketsEngine object with observer aConsole, using two-phase construction. Return pointer to object. static CSocketsEngine* NewL(MUINotify& aConsole)
NewLC	Create CSocketsEngine object with observer aConsole, using two-phase construction. Push the object onto the cleanup stack. Returns pointer to object. static CSocketsEngine* NewLC(MUINotify& aConsole)
Destructor	Destroy object and all resources owned by it. ~CSocketsEngine();
ConstructL	Perform second phase construction of the CSocketsEngine. This includes creating objects for sockets reading and writing. void ConstructL();
Socket Connect	First stage of establishing socket connection. It handles establishing connection with IP address, or domain name. Performs lookup procedure in case of domain name. void ConnectL() Establish connection directly with IP address. void ConnectL(TUInt32 aAddr) Performs the connection procedure in synchronous fashion. void Sync_Connect()
Connected	Returns the state of socket connection. TBool Connected() const;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsEngine : public MTimeoutNotify

Sync_LookupL	Performs domain name lookup in a synchronous fashion. void Sync_LookupL();
Socket Disconnect	Closes connection, and performs cleanup procedure. void Disconnect(); Performs the disconnection procedure in synchronous fashion. void sync_Disconnect();
SetServerName	Set server name with aName. void SetServerName(const TDesC& aName);
ServerName	Returns the server name. const TDesC& ServerName() const;
SetPort	Set the port number with aPort. void SetPort(TInt aPort);
Port	Return the port number. TInt Port() const;
Read	Read sockets receive buffer void Read();
WriteL	Write aData string to send buffer void WriteL(const TDesC& aData);
Cancel	Cancel outstanding socket requests. Perform cleanup procedure. void Cancel();
enum TSocketsEngineState	States describing the different stages of socket communications. ENotConnected, EConnecting, EConnected,

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsEngine : public MTimeoutNotify

	ETimedOut, ELookingUp, ELookUpFailed, EConnectFailed, EDisconnecting
ChangeStatus	Change socket status, iEngineStatus. Prints state change to log file as well. void ChangeStatus(TSocketsEngineState aNewStatus);
Print	Print debug information to log file. void Print(const TDesC& aDes);
RecvMessageNoBlockL	Wrapper function to access the receive shadow buffer. TUint RecvMessageL(char *aMessage, unsigned int aLength); TUint RecvMessageNoBlockL(char *aMessage, unsigned int aLength);
ResetRecvBuffer	Reset receive shadow buffer void ResetRecvBuffer();
iEngineStatus	Object current status TSocketsEngineState iEngineStatus
iConsole	Observer to print messages MUINotify& iConsole
iSocketsRead	Socket reader object CSocketsRead* iSocketsRead
iSocketsWrite	Socket writer object CSocketsWrite* iSocketsWrite
iSocket	Socket resource RSocket iSocket

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsEngine : public MTimeoutNotify

iSocketServ	Server socket resource.
	RSocketServ iSocketServ
iResolver	DNS name resolver
	RHostResolver iResolver
iNameEntry	DNS Lookup result.
	TNameEntry iNameEntry;
iNameRecord	DNS Lookup result.
	TNameRecord iNameRecord;
iTimer	Timer active object
	CTimeoutTimer* iTimer
iAddress	Server address
	TInetAddr iAddress;
iPort	Port number for connect.
	TInt iPort
iServerName	Server name
	TBuf<KMaxServerNameLength> iServerName
IStatus	Local Status tracking variable
	TRequestStatus iStatus;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4.3 CSocketsWrite

class CSocketsWrite : public MTimeOutNotify

An instance of class CSocketsWrite is an object responsible for sending data through the Symbian sockets module to the server.

Constructor	Creates CSocketsWrite object with observer aConsole and resource aSocket. Using two-phase construction. CSocketsWrite(MUINotify& aConsole, RSocket& aSocket);
NewL	Create CSocketsWrite object with observer aConsole, socket resource aSocket, using two-phase construction. Return pointer to object. static CSocketsWrite* NewL(MUINotify& aConsole, RSocket& aSocket);
NewLC	Create CSocketsWrite object with observer aConsole, socket resource aSocket, using two-phase construction. Push object onto cleanup stack. Return pointer to object. static CSocketsWrite* NewLC(MUINotify& aConsole, RSocket& aSocket);
ConstructL	Perform second phase construction of the CSocketsWrite. This includes initializing timer and write socket state. void ConstructL();
Destructor	Destory object and all resources owned by it. ~CSocketsWrite();
IssueWriteL	Checks for socket status and send buffer condition for validity. Prepare populate the send buffer with data. Calls SendNextPacket to actually send data through socket. void IssueWriteL(const TDesC8& aData);
Cancel	Initiates cancel procedure. Terminates timer object void Cancel();
TimerExpired	Checks for timer expiration, and issue message accordingly. void TimerExpired();
SendNextPacket	Writes the send buffer to socket.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsWrite : public MTimeOutNotify	
	void SendNextPacket();
Sync_SendNextPacket	Perfomes synchronous data sending procedure. void Sync_SendNextPacket();
TWriteState	State of the Write socket. enum TWriteState {ESending, EWaiting ,ECommsFailed};
KWriteBufferSize	Size of write buffer. The guideline for this is defined by the protocol requirements.
iSocket	Socket resource RSocket& iSocket;
iConsole	Observer object for displaying and logging messages for socket writing. MUINotify& iConsole;
iTransferBuffer	Accumulate data to send in here TBuf8<KWriteBufferSize> iTransferBuffer
iWriteBuffer	Holds data currently being sent to socket TBuf8<KWriteBufferSize> iWriteBuffer
iTimer	Timer object CTimeoutTimer* iTimer;
iTimeOut	Define limits for time out condition Tint iTimeOut;
iWriteStatus	Holds the Socket write state. TwriteState iWriteStatus
iStatus	Holds system status for write socket TrequestStatus iStatus

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4.4 CSocketsRead

class CSocketsRead : public MTimeoutNotify

An instance of class CSocketsRead is an object responsible for receiving data through the Symbian Sockets module from the server

Constructor	Creates CSocketsRead object with observer aConsole and resource aSocket. Using two-phase construction. CSocketsRead(MUINotify& aConsole, RSocket& aSocket);
NewL	Create CSocketsRead object with observer aConsole, socket resource aSocket, using two-phase construction. Return pointer to object. static CSocketsRead* NewL(MUINotify& aConsole, RSocket& aSocket);
NewLC	Create CSocketsRead object with observer aConsole, socket resource aSocket, using two-phase construction. Push object onto cleanup stack. Return pointer to object. static CSocketsRead* NewLC(MUINotify& aConsole, RSocket& aSocket);
Destructor	Destory object and all resources owned by it. ~CSocketsRead();
ConstructL	Perform second phase construction of the CSocketsWrite. This includes initializing timer and write socket state. void ConstructL();
Cancel	Initiates cancel procedure. Terminates timer object void Cancel();
Start	Initiate Socket reading. void Start();
TimerExpired	Checks for timer expiration, and issue message accordingly. void TimerExpired()
IssueReadL	Perform Socket read and store data into read buffers.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsRead : public MTimeOutNotify

	void IssueReadL()
ReadCompletedL	Transfer read data from receive buffer into shadow buffer void ReadCompletedL(TDesC8 &aBuffer, TInt aLength);
ResetRecvBuffer	Reset shadow buffer start and end pointers void ResetRecvBuffer();
RecvMessageL	Download agent core request for data from shadow buffer. Returns aLength data from Shadow buffer. TUint RecvMessageL(char *aMessage, unsigned int aLength);
RecvMessageNoBlockL	Download agent core request for data from shadow buffer. Returns as many bytes available in shadow buffer up to length aLength TUint RecvMessageNoBlockL(char *aMessage, unsigned int aLength);
KReadBufferSize	Size of write buffer. enum { KReadBufferSize = 4096 };
EReadState	Read Socket states. enum EReadState {EReading, EReadDone, EReadError};
iSocket	Socket to read data from RSocket& iSocket
iConsole	Observer object for displaying and logging messages for socket reading. MUINotify& iConsole
iBuffer	Buffer for receiving data TBuf8<KReadBufferSize> iBuffer
iDummyLength	Returns length of data read. TSockXfrLength iDummyLength
iRecvBuffer	Shadow circular buffer to accumulate received data for download agent use.

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

class CSocketsRead : public MTimeoutNotify

	TBuf8<KBufferSize> iRecvBuffer;
iRecvBuffer_begin	Shadow buffer data starts address. TInt iRecvBuffer_begin
iRecvBuffer_end	Shadow buffer data end address. TInt iRecvBuffer_end
iTimer	Timer object CTimeoutTimer* iTimer;
iTimeout	Define limits for time out condition TInt iTimeout;
iReadState	Read Socket status EReadState iReadState;
iStatus	Holds system status for read socket TRequestStatus iStatus;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4.5 CTimeoutTimer

class CTimeoutTimer : public CTimer

An instance of class CTimeoutTimer is an object that notifies other objects of elapsed time.

NewL	Constructs a CTimeoutTimer object with apriority and observer aTimeoutNotify using two phase construction. Return pointer to the object. static CTimeoutTimer* NewL(const TInt aPriority, MTimeoutNotify& aTimeoutNotify);
NewLC	Constructs a CTimeoutTimer object with apriority and observer aTimeoutNotify using two phase construction. Pushes the object onto the cleanup stack. Returns pointer to the object. static CTimeoutTimer* NewLC(const TInt aPriority, MTimeoutNotify& aTimeoutNotify);
Destructor	Destroys object and releases all resources owned by it. ~CTimeOutTimer();
RunL	Service the active object when iStatus is set to pending. virtual void RunL();
Constructor	Constructs CtimeOutTimer with apriority and observer aTimeOutNotify using two-phase construction. CTimeOutTimer(const TInt aPriority, MTimeOutNotify& aTimeOutNotify);
ConstructL	Performs second phase construction. void ConstructL();
iNotify	Reference an MtimeOutNotify object. MTimeOutNotify& iNotify;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.

5.4.6 MTimeOutNotify

class MtimeOutNotify

An instance of class MtimeOutNotify is an object which implements the timeout function

TimerExpired virtual void TimerExpired() = 0;

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone.

© 2002 Bitfone Corporation.